

A Fast Two Pass Multi-Value Segmentation Algorithm based on Connected Component Analysis

Dibyendu Mukherjee
University of Windsor
401 Sunset Avenue, Windsor, Canada
dibyendu.mukherjee@ieee.org

Abstract

Connected component analysis (CCA) has been heavily used to label binary images and classify segments. However, it has not been well-exploited to segment multi-valued natural images. This work proposes a novel multi-value segmentation algorithm that utilizes CCA to segment color images. A user defined distance measure is incorporated in the proposed modified CCA to identify and segment similar image regions. The raw output of the algorithm consists of distinctly labelled segmented regions. The proposed algorithm has a unique design architecture that provides several benefits: 1) it can be used to segment any multi-channel multi-valued image; 2) the distance measure/segmentation criteria can be application-specific and 3) an absolute linear-time implementation allows easy extension for real-time video segmentation. Experimental demonstrations of the aforesaid benefits are presented along with the comparison results on multiple datasets with current benchmark algorithms. A number of possible application areas are also identified and results on real-time video segmentation has been presented to show the promise of the proposed method.

1. Introduction

Connected components analysis is a well-explored fundamental topic in image & video processing. In binary images, CCA is used to segment 4 or 8-connected regions by assigning unique labels. The classical approaches for CCA date back to the 1960s [18, 19] and mainly focussed on labeling binary images. These labels can be used for shape characterizations to object recognition. Even today, binary CCA is very popular for object detection, tracking, segmentation and other computer vision applications. With the diversity of applications, methods for CCA have also improved over time [10, 11, 22, 24]. An overview of the advancements is provided in [10]. Although the improve-

ments in the area of CCA is worth noticing, very few of the



Figure 1: Conditional segmentation results of the proposed method.

approaches had their focus on labeling connected components in color images. Binary images consist of two classes of values: 0 and 1. It is simpler to treat the pixels to be part of a region (value of 1) or no region (value of 0) and segment accordingly. However, for color images, the problem of labeling is more complicated due to the followings: 1) each pixel belongs to a region; 2) the regions are not unique in terms of color. As color values are randomly distributed in a natural image, it is not possible to classify a region in terms of a single color value, without specific criteria. There have been a few approaches for color image segmentation based on CCA [3, 14, 15, 20]. However, most of them are limited to constant color regions and higher computational complexity. This work proposes a linear-time two-pass CCA based multi-valued image segmentation algorithm, and provides different measures for classification of a color region in terms of color values, Gradient values and Saliency values. The algorithm is based on similarity of neighboring pixels. The architecture is influenced from the two-pass binary CCA proposed by Haralick [12]. Haralick's algorithm has a space efficient run-length implementation. This implementation has been partially adopted with significant changes: : 1) The algorithm has been introduced for multi-valued image data; 2) unlike Haralick's algorithm, the initialization of data structures are carried out in the top-down pass. So, there are no hidden computational costs involved; 3) the algorithm has a modular architecture in which, the segmentation criteria is called from outside the original function flow signifying a completely user-specific criteria that can be chosen by a run-time polymorphism in any advanced computer language that supports it; 4) The

algorithm does not require the number of clusters as input, and 5) the implementation renders it suitable for real-time applications or streaming videos.

The algorithm is explained in Section 2, followed by a discussion on the distance measures for segmentation 3. The experimental results are reported in Section 4. Finally, the paper is concluded in Section 5.

2. Algorithm

A number of important definitions are required before the explanation of the proposed algorithm. The definitions are provided with respect to an image IM of N_R rows, N_C columns and N_{CH} number of color channels:

Similarity: Two pixels PX_p and PX_q at locations (r_p, c_p) and (r_q, c_q) respectively, are termed *similar* if they share a *common property*. This *common property* can be defined in terms of color similarity, gradient orientation similarity or any other similarity or distance measurement operation, depending on the application.

Run: A run is defined by a set of contiguous and *similar* pixels, in a particular row. It is uniquely represented by the row index, and its starting and ending column index.

Equivalence: Two runs p and q are equivalent if they satisfy at least one of the following criteria:

1. TYPE-I: p and q share a common boundary and any two pixels PX_p and PX_q associated with p and q respectively, are *similar* using the *common property* criteria.
2. TYPE-II: p and q are equivalent to runs u and v respectively, using criteria 1. If u and v are proved equivalent, p and q are also equivalent.

Label: Apart from its index, a run also has a permanent label. When two runs are equivalent, they are merged by assigning them a single label. In a segmented image, all equivalent runs are assigned same labels.

In this work, *similarity* is verified using a function $dist(r_p, c_p, r_q, c_q)$ that takes pixel coordinate pairs (r_p, c_p) and (r_q, c_q) as input, and returns true, if PX_p and PX_q are *similar*. Otherwise, it returns false. There can be several runs in a row. A run ends when its end pixel is not *similar* to its subsequent pixel in the same row, or if the row ends. An example image is shown in Figure 2. The first row has first 4 elements of red color, the next single element of blue color, followed by 3 more elements in red and the last two elements in green. Hence, there are four runs in the row: red (1-4), blue (5-5), red (6-8) and green (9-10). Considering $dist()$ to have a threshold (typically 10-15) on Euclidean distance between two color pixels for this example, all “reddish” pixels are considered as “Red” and similar for Blue and Green. Thus, second row has 6 runs while third row has 3 runs.

The algorithm has two passes. In the top-down pass, the runs are computed, labels are assigned, and equivalences are established. In the bottom-up pass, equivalent runs are assigned same labels. Hence, the image is segmented. In the following paragraphs, the two passes are discussed in details with related data structures. Finally, the pseudo-codes of the algorithm are presented using three procedures: SEGMENT 1, INITLABEL 2, MAKEEQUIVALENT 3.

In the top-down pass, the runs are computed and their information are populated in two data structures: *rowBlock* and *colBlock*. *rowBlock* has a dimension of $N_R \times 2$ and keeps the number of runs present in each row. The two elements *firstRun* and *lastRun* of i^{th} row in *rowBlock* keep the index of the first run and last run in i^{th} row of IM , respectively. In the example image, the first row has four runs indexed 1 to 4. Thus, $rowBlock(1).firstRun = 1$, $rowBlock(1).lastRun = 4$. Here, $rowBlock(i)$ represent the i^{th} row, and the “.” operator is used to refer to the elements of that row.

										rowBlock	firstRun	lastRun
										1	1	4
										2	5	10
										3	11	13

colBlock	row	colStart	colEnd	permLabel (initial)	permLabel (final)
1	1	1	4	1	1
2	1	5	5	2	2
3	1	6	8	3	1
4	1	9	10	4	4
5	2	1	3	1	1
6	2	4	4	2	2
7	2	5	6	1	1
8	2	7	7	5	2
9	2	8	9	3	1
10	2	10	10	4	4
11	3	1	4	1	1
12	3	5	6	2	2
13	3	7	10	4	4

eqClass	eqLabel	labelNext	label	next
1	3	1	1	0
2	2	2	2	5
3	0	3	1	1
4	4	4	4	0
		5	2	0

Segmented output												
1	1	1	1	2	1	1	1	4	4			
1	1	1	2	1	1	2	1	1	4			
1	1	1	1	2	2	4	4	4	4			

Figure 2: The data structures for a sample run on an image. From top-left to bottom-right: the image, *rowBlock*, *colBlock*, *eqClass*, *labelNext* and segmented output.

colBlock has the dimension of $N_{RN} \times N_C + 4$ with N_{RN} representing the number of runs. i^{th} row represents the necessary information for i^{th} run. The elements *row*, *colStart*, *colEnd* and *permLabel* in i^{th} row of *colBlock*, represent the row index, start and end column index, and permanent label for the i^{th} run. In the top-down pass for p^{th} run in r^{th} row, every pixel PX_p is compared with its three neighboring pixels in $(r-1)^{\text{th}}$ row (if the row exists), using the $dist()$ function. If a neighboring pixel PX_q belonging to run q is similar to PX_p , the runs p and q are TYPE-I equivalent. Thus, they are made equivalent using a procedure MAKEEQUIVALENT 3. Finally, the *permLabels* are

replaced with the equivalent labels in the bottom-up pass as provided in procedure SEGMENT 1. An image and its sample run information are provided in Figure 2. The initial labels after top-down pass, and final labels after bottom-up pass are provided in *colBlock* as well.

Finally, TYPE-II equivalences are resolved using the two data structures: *eqClass* and *labelNext*. If runs p and q with *permLabels* L_p and L_q respectively, are equivalent, they must belong to same *class* c . A *class* c has a *unique* label L_c , and the *permLabels* of p and q are made equivalent by associating both of them to L_c . The *labelNext* structure has two elements: *label* and *next*. If p already belongs to *class* c , *labelNext*(L_p).*label* = L_c ($\neq 0$). If p and q need to be made equivalent, L_q needs to be associated to L_c . If q does not belong to any *class* i.e. *labelNext*(L_q).*label* = 0), the procedure is easy. However, if q belongs to *class* \hat{c} i.e. *labelNext*(L_q).*label* = $L_{\hat{c}}$ ($\neq 0 \neq L_c$), c and \hat{c} should be equivalent. This equivalence is kept in data structure *eqClass* element *eqLabel*. Thus, in this case, *labelNext*(L_q).*label* is made equal to L_c and *eqClass*(L_c) is made equal to $L_{\hat{c}}$ to link both the classes. This process is established through the procedure MAKEEQUIVALENT 3 depicted in the algorithm.

3. The Distance Measure

The distance measure is used to identify TYPE-I equivalence of two runs based on neighboring pixel relationships, and directly controls the quality of segmentation. Based on the relationships between two neighboring pixels, they can be part of same segment or different segment. A bare format of the function is as follows:

```

1: function dist( $r_1, c_1, r_2, c_2$ )
2:   similar = false;
3:   if  $r_2 > 0$  then
4:     similar = (custom logic to validate similarity)
5:   end if
6:   RETURN similar
7: end function

```

In the example Figure 2, an example distance measure is as follows:

$$similar = \|IM(r_1, c_1) - IM(r_2, c_2)\| < Th, \quad (1)$$

Where, $IM(R_1, c_1)$ represent the color information of pixel (r_1, c_1). $\|\cdot\|$ represents the Euclidean norm, and Th is a distance threshold, typically ranging from 0 – 50 for a pixel value range of 0 – 255. Thus, *similar* is true for pixels with a small color difference. For Figure 2, Th normally lies in the range of 5 – 10. With $Th = 0$, the pixels in a neighborhood sharing equal color values would be segmented in a distinct region.

Based on this idea, there may be a large number of possible distance measures. If only reddish pixels are needed to be put together, both pixels can be compared to Red and a joint threshold can be put. Similarly, edge information can

be included to segment regions separated by edges. However, due to limited space, a discussion on only three types of distance measure is provided. The first one is the Euclidean distance already mentioned in Eq. 1. The other two types discussed are: Gradient based distance measure, and Saliency based distance measure.

3.1. Gradient based Distance Measure

Gradient I_G of an image is the second norm of the partial differentials I_X and I_Y of the image. Partial differentials can be computed by convolving the image with differential filters along X and Y directions, e.g. Sobel, Prewitt etc. Qualitatively, I_G provides the edge information of the image and it contains the magnitude of the differentiation. In this work, simple differential filters, two basic differential filters $H_X = [-1, 0, 1]$ and $H_Y = H_X^T$ are used to provide the differentials along X and Y directions, respectively. Here, T denotes the transpose. Mathematically,

$$I_X = I * H_X; I_Y = I * H_Y; I_G = \sqrt{I_X^2 + I_Y^2}. \quad (2)$$

Here, $*$ denotes convolution. The Gradient information can be used many ways to make a distance measure. However, in this work, individual thresholdings are used:

$$similar = (I_G(r_1, c_1) < Th) \& (I_G(r_2, c_2) < Th). \quad (3)$$

Equation 3 is based on the nature of Gradient information. I_G has low value in relatively uniform regions while having a high value in boundary areas. Two pixels are not *similar* if at least one of them reside on edges having a high Gradient value. The difference between simple Euclidean distance based segmentation and Gradient information based segmentation has been demonstrated in the experiments Section 4.

3.2. Saliency based Distance Measure

Saliency is based on Human Visual System (HVS) and tries to provide a contrast between regions based on their importance to HVS. A foreground having distinctive features can be more salient to the eyes of an observer compared to a relatively uniform background. There have been many algorithms to find salient regions of an image. Considering the execution performance of the methods, the work by Achanta *et al.* [1] is found to be well-suited for this work. To reduce duplication, the theory of the method is not presented in this work. Interested readers are encouraged to go through the reference for details. For now, it would suffice to say that the method takes an RGB color image as input, converts it to CIE Lab space, and provides the saliency map S containing Saliency value of each pixel as the Euclidean distance between the pixel's Lab value and the mean Lab.

Algorithm 1 Two-Pass Segmentation Algorithm

```
1: procedure SEGMENT
2:    $cr = 0$ ;
3:    $cl = 0$ ;
4:   for  $r = 1, N_R$  do                                     ▷ Top-down pass
5:      $fr = \text{false}$ ;
6:      $cx = -1$ ;  $cy = -1$ ;
7:     for  $c = 1, N_C$  do
8:        $matched = \text{dist}(r, c, cx, cy)$ ;
9:       if ( $fr$ ) then
10:        if  $matched$  then
11:           $cx = c$ ;  $cy = r$ ;
12:        else
13:           $fr = \text{false}$ ;
14:        end if
15:      end if
16:      if ( $fr = \text{false}$ ) & ( $matched = \text{false}$ ) then
17:        if ( $cr > 0$ ) then
18:          if  $colBlock(cr).label == 0$  then
19:             $cl = cl + 1$ ;
20:             $colBlock(cr).permLabel = cl$ ;
21:          end if
22:        end if
23:         $cr = cr + 1$ ;
24:         $cx = c$ ;  $cy = r$ ;
25:         $colBlock(cr).row = r$ ;
26:         $colBlock(cr).colStart = c$ ;
27:         $colBlock(cr).permLabel = 0$ ;
28:         $fr = \text{true}$ ;
29:        if  $rowBlock(r).firstRun == 0$  then
30:           $rowBlock(r).firstRun = cr$ ;
31:        end if
32:         $rowBlock(r).lastRun = cr$ ;
33:      end if
34:      if ( $r > 1$ ) then
35:         $INITLABEL(r, c)$ ;
36:      end if
37:      if  $fr$  then
38:         $colBlock(cr).colEnd = c$ ;
39:      end if
40:       $idxImg(r, c) = cr$ ;
41:    end for
42:  end for
43:  if ( $colBlock(cr).permLabel == 0$ ) then
44:     $cLabel = cLabel + 1$ ;
45:     $colBlock(cr).permLabel = cLabel$ ;
46:  end if
47:  for  $r = imSize(1), -1, 1$  do                               ▷ Bottom-up pass
48:     $p = rowBlock(r).firstRun$ ;
49:     $pLast = rowBlock(r).lastRun$ ;
50:    if ( $p \neq 0$ ) then
51:      while ( $p \leq pLast$ ) do
52:         $pl = colBlock(p).permLabel$ ;
53:         $ql = labelNext(pl).label$ ;
54:        if  $ql \neq 0$  then
55:           $colBlock(p).permLabel = ql$ ;
56:        end if
57:         $p = p + 1$ ;
58:      end while
59:    end if
60:  end for
61: end procedure
```

Algorithm 2 Initial Labeling

```
1: procedure INITLABEL( $r, c, cr$ )
2:    $matched = \text{dist}(r, c, r - 1, c)$ ;
3:   if  $matched$  then
4:      $tr = idxImg(r - 1, c)$ ;
5:      $pl = colBlock(cr).permLabel$ ;
6:      $ql = colBlock(tr).permLabel$ ;
7:     if  $pl == 0$  then
8:        $colBlock(cr).permLabel = ql$ ;
9:     else
10:       $MAKEEQUIVALENT(pl, ql)$ ;
11:    end if
12:  end if
13:  if ( $c > 1$ ) then
14:     $matched = \text{dist}(r, c, r - 1, c - 1)$ ;
15:    if  $matched$  then
16:       $tr = idxImg(r - 1, c - 1)$ ;
17:       $pl = colBlock(cr).permLabel$ ;
18:       $ql = colBlock(tr).permLabel$ ;
19:      if  $pl == 0$  then
20:         $colBlock(cr).permLabel = ql$ ;
21:      else
22:         $MAKEEQUIVALENT(pl, ql)$ ;
23:      end if
24:    end if
25:  end if
26:  if ( $c < imSize(2)$ ) then
27:     $matched = \text{dist}(r, c, r - 1, c + 1)$ ;
28:    if  $matched$  then
29:       $tr = idxImg(r - 1, c + 1)$ ;
30:       $pl = colBlock(cr).permLabel$ ;
31:       $ql = colBlock(tr).permLabel$ ;
32:      if  $pl == 0$  then
33:         $colBlock(cr).permLabel = ql$ ;
34:      else
35:         $MAKEEQUIVALENT(pl, ql)$ ;
36:      end if
37:    end if
38:  end if
39: end procedure
```

The distance measure can be derived in a number of ways depending on the application and implementation. However, the main focus of this example is to show how the saliency information can be used to segment the salient regions or the non-salient regions of the image. To properly segment the salient regions, the distance measure is as follows:

```
1:  $Sm = \text{mean}(S)$ ;
2: if ( $S(r_1, c_1) < Sm$ ) & ( $S(r_2, c_2) < Sm$ ) then
3:    $similar = \|IM(r_1, c_1) - IM(r_2, c_2)\| < Th2$ 
4: else
5:    $similar = \|IM(r_1, c_1) - IM(r_2, c_2)\| < Th$ 
```

6: **end if**

Here, Sm denotes the mean of the Saliency image S . $Th2$ is a low threshold typically $[1/5 - 1/10]^{\text{th}}$ of Th . The conditional statements have a purpose. Generally, each segment is treated as a separate *object*. If both pixels have Saliency below a margin (Sm), they belong to non-salient objects and need less attention for segmentation. A low threshold $Th2$ is used as number of segments or objects are less important. If otherwise, at least one pixel is salient, a higher threshold Th is used to loosen the constraint of color

Algorithm 3 Make Two Labels Equivalent

```
1: procedure MAKEEQUIVALENT(pl, ql)
2:   lpl = labelNext(pl).label;
3:   lql = labelNext(ql).label;
4:   if (lpl == 0) & (lql == 0) then
5:     labelNext(pl).label = pl;
6:     labelNext(ql).label = pl;
7:     labelNext(pl).next = ql;
8:     labelNext(ql).next = 0;
9:     eqClass(pl).eqLabel = pl;
10:  else if lpl == lql then
11:    Do nothing
12:  else if (lpl ≠ 0) & (lql == 0) then
13:    bgn = lpl;
14:    label(ql).label = bgn;
15:    label(ql).next = eqClass(bgn).eqLabel;
16:    eqClass(bgn).eqLabel = ql;
17:  else if (lql ≠ 0) & (lpl == 0) then
18:    bgn = lql;
19:    label(pl).label = bgn;
20:    label(pl).next = eqClass(bgn).eqLabel;
21:    eqClass(bgn).eqLabel = pl;
22:  else if (lql ≠ 0) & (lpl ≠ 0) then
23:    bgn = lql;
24:    member = eqClass(bgn).eqLabel;
25:    eql = lpl;
26:    while label(member).next ≠ 0 do
27:      label(member).label = eql;
28:      member = label(member).next;
29:    end while
30:    label(member).label = eql;
31:    label(member).next = eqClass(eql).eqLabel;
32:    eqClass(eql).eqLabel = eqClass(bgn).eqLabel;
33:    eqClass(bgn).eqLabel = 0;
34:  end if
35: end procedure
```

when segmenting, so that if the pixels belong to same salient object but have distant colors, they can still be merged into one object.

To segment non-salient regions instead, the condition in line 3 can simply be changed as: $(S(r_1, c_1) > Sm) \text{OR} (S(r_2, c_2) > Sm)$. The effect of changing the condition is shown in Figure 3. In the middle image, the salient region is highly segmented using condition in line 3. But, in the right-most image, with a reverse condition, salient region has more textures that are not segmented. More results are provided in Section 4.



Figure 3: Saliency effect: left - original image, middle - salient segmentation, right - nonsalient segmentation

4. Experimental Results

The algorithm has been coded in Java and a GUI has been made for ease of use. Java has been chosen mainly because, it has the capability of run-time polymorphism while having a fast execution like C. Using polymorphism, we can define as many definitions of *dist()* as we need, and choose to call whichever required in run-time. In this section, the three distance measures defined in Section 3 have been used and several types of results are provided. To properly address the proposed method with different distance measures, the following nomenclature has been followed: Two pass segmentation method (tps) with 1) Euclidean distance measure - tpsEuclid, 2) Gradient distance measure - tpsGradient and 3) Saliency based measure - tpsSaliency. Image segmentation results are reported on the Berkeley Segmentation databases: BSDS300 [16] (with 300 images) and

BSDS500 [2] (with 500 images) consisting of natural images from various scene categories. Each image has been segmented by several human subjects. Thus, there exist multiple groundtruths for each image.

Both qualitative and quantitative results are reported in this work. For quantitative results, we used the common criteria used to compute segmentation errors: 1) Probabilistic Rand Index (PRI) - measures the likelihood of a pixel pair being grouped consistently in two segmentations, 2) Variation of Information (VoI) - computes the amount of information in one result not part of the other one, and 3) Global Consistency Error (GCE) - measures the extent to which one segmentation is a refinement of the other one. When compared to groundtruth, a higher value for PRI and lower values for VoI and GCE denote better results. As there are multiple groundtruths, the results are averaged over all the groundtruths for each image.

The results are divided into separate subsections according to their objectives. Section 4.1 provides a qualitative and quantitative comparison among the three proposed distance measures on BSDS500 with different threshold values. Section 4.2 is dedicated to quantitative comparison of the proposed method with some of the benchmark algorithms from current literatures. An estimation of execution time has been provided in Section 4.3. Finally, possible applications are discussed in Section 4.4.

4.1. Comparison Among Different Distance Measures

Different distance measures produce distinct segmentations on same image. However, the quality of segmentation also varies greatly with the distance threshold used. In this section, the three methods tpsEuclid, tpsGradient and tpsSaliency have been run on the 500 images of BSDS500 for distance thresholds 5, 10, 15, 20, 25 and 30. The graph in Figure 4 presents the variation of PRI with the varia-

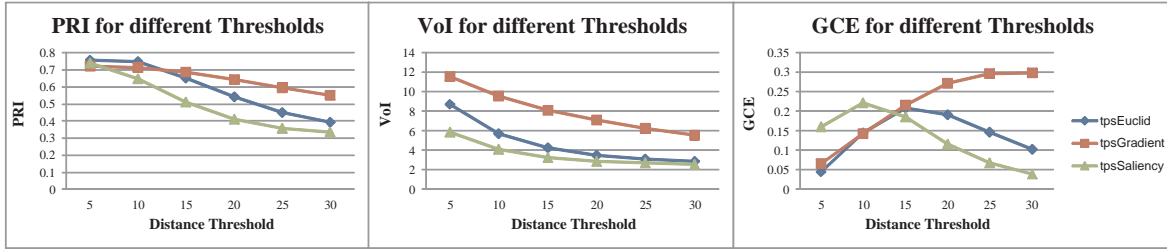


Figure 4: Variation of PRI, VoI and GCE with variation in distance thresholds.



Figure 5: Qualitative results: First column - original image, second and third columns represent tpsEuclid results with distance threshold 5 and 10, respectively. Similarly, fourth and fifth columns represent tpsGradient and sixth and seventh columns represent tpsSaliency for thresholds 5 and 10, respectively.

tion of distance threshold used for each method. With increase of threshold, distinctly different segments are pro-

gressively merged yielding lower quality for PR. However, lower amount of segments also reduce VoI. GCE, on the other hand, depends more on the mapping of segments

rather than number of segments, and follows a separate trend.

Some qualitative results are shown in Figure 5. According to the images, for different image content, different distances are more suitable. Specifically, images with high salient regions can be better segmented by tpsSaliency, whereas images with higher number of distinct segments are better suited for tpsEuclid and tpsGradient.

4.2. Benchmark Comparison Results

In this section, quantitative comparisons are carried out with some of the benchmark algorithms from current literature. The following algorithms are tested: Ncut [21], MShift [4], FH [8], JSEG [6], Multi-scale Ncut (MN-cut) [5], Normalized Tree Partitioning (NTP) [23], Saliency Driven Total Variation (SDTV) [7], Texture and Boundary Encoding-based Segmentation (TBES) [17] and Segmentation by Aggregating Superpixels (SAS) [13]. The scores of the algorithms are collected from [13].

Table 1: Quantitative comparison of proposed method with other methods

Methods	PRI	VoI	GCE
Ncut [21]	0.7242	2.9061	0.2232
MShift [4]	0.7958	1.9725	0.1888
FH [8]	0.7139	3.3949	0.1746
JSEG [6]	0.7756	2.3217	0.1989
MNcut [5]	0.7559	2.4701	0.1925
NTP [23]	0.7521	2.4954	0.2373
SDTV [7]	0.7758	1.8165	0.1768
TBES [17]	0.80	1.76	N/A
SAS [13]	0.8319	1.6849	0.1779
tpsEuclid	0.7602	8.6167	0.0446
tpsGradient	0.7129	11.5739	0.0691
tpsSaliency	0.7359	5.8860	0.1576

The quality of results entirely depends on $dist()$ and may improve with other distance measures. Also, the application areas of the proposed approach are not limited to image segmentation due to its uniqueness of cluster independence, application-specific distance measure and real-time applicability. Regarding real-time applicability, the proposed method is much faster compared to any other algorithm (reported rates of the benchmark algorithms are at least more than 5 seconds/image of size 481×321). An estimation of time complexity has been provided in the next section.

4.3. Execution Time Estimation

In this section, average execution times are tabulated for different image sizes. The execution time of the two-pass algorithm depends on the number of runs. With lower value of runs, the passes over each row takes lower time as each pass loops over each run in a row. Thus, the maximum number of iteration arises when, number of runs equals the number of pixels and the algorithm iterates over each pixel twice (top-down and bottom-up) achieving an absolute $O(n)$ complexity. To achieve this extreme condition, a $dist()$ function is used to always return false. As this does not depend on

image content any more, an image with random pixel values is used. The original image size is $4096 \times 4096 \times 3$, and it is reduced dyadically in size to as small as $128 \times 128 \times 3$. The times taken by the two-pass algorithm are tabulated in Table 2 in milliseconds (second & third image dimensions are not shown). However, this does not include the time taken to compute the mean of each segment and display the image on a computer screen or save it, as these are not part of the original algorithm. For the execution, a desktop computer with 3 GHz AMD Phenom II X6 Processor is used.

Referring to the table, if the execution times are plotted on a graph against the numbers of pixels, it will be an approximation of a straight line. This signifies the linear time complexity of the algorithm.

Table 2: Execution Time Estimation

Size	4096	2048	1024	512	256	≤ 128
Time (ms)	755	70	16	5	1	< 1

4.4. Applications



Figure 6: Detection and tracking: First, second, third & fourth columns shows the original frames, segmented water, segmented boat, and segmented non-water regions, respectively.

The proposed method has several applications. It can be used as a preprocessing step for any image or video based algorithm like conditional segmentation (e.g. Figure 1), motion or event detection, and tracking. One of the main advantages of using the algorithm is that the outputs contain connected and labeled regions along with the segmentation. This can help in spatiotemporal tracking, detection of specific objects and real-time video segmentation. A number of preliminary results are shown in Figure 6, on the Canoe video sequence from the Change-Detection datasets [9]. The results are obtained by simply changing the distance measure. The second column is obtained by comparing the blue channel of each frame to moderate blue (RGB: 0,0,220) with threshold of 72. Third column is obtained in a similar way by comparing with any reddish pixel of the Canoe (example RGB: 140, 65, 90) with threshold 72. In both cases, distance value must be lower than 72 to be *similar*. Fourth column demonstrates the dual of the criteria for second column: threshold on blue channel but distance value must be higher than 72.

It is a legitimate assumption that the scene content does not change abruptly, for two subsequent frames of a video. Thus, based on the similarity of pixels in a region, a segmented region should have similar mean color in two subsequent frames. This considerably reduces the flickering of colors in a video segmentation. A number of subsequent frames are shown in Figure 7 for a threshold of 15 on tpsEuclid. Although this is a preliminary application, better segmentation quality may be achieved with different distance measures.



Figure 7: Video segmentation: First row shows four frames 910, 930, 950 and 970, respectively. Second row shows corresponding segmentations.

5. Conclusion

In this work, we proposed a novel method for multi-value segmentation based on connected components analysis. Primarily, the method enables us to group similar regions in multi-valued image. The similarity criteria for segmentation can be custom defined. The efficacy of the method has been demonstrated by its use on real-time image and video segmentation. It is worth noticing, that there is no need of any seed or number of clusters for the process of segmentation. However, one of the drawbacks lies in defining the appropriate distance measure for a particular application. The distance measures applied in the work have a high dependency on texture variations. Thus, they fail in high-textured regions and are prone to segment leakage. Thus, future advancements would include searching for a distance measure relatively insensitive to high texture variations.

References

- [1] R. Achanta, S. Hemami, F. Estrada, and S. Susstrunk. Frequency-tuned salient region detection. In *Proc. of IEEE Conf. on Comp. Vis. and Pat. Rec.*, pages 1597–1604, 2009. 3
- [2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. on Pat. Anal. and Mach. Intelli.*, 33(5):898–916, 2011. 5
- [3] M. E. Celebi. A simple and efficient algorithm for connected component labeling in color images. *Proc. of SPIE*, 8295:82951H–82951H–6, 2012. 1
- [4] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Trans. on Pat. Anal. and Mach. Intelli.*, 24(5):603–619, 2002. 7
- [5] T. Cour, F. Benezit, and J. Shi. Spectral segmentation with multiscale graph decomposition. In *Proc. of IEEE Conf. on Comp. Vis. and Pat. Rec.*, volume 2, pages 1124–1131 vol. 2, 2005. 7
- [6] Y. Deng and B. Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE Trans. on Pat. Anal. and Mach. Intelli.*, 23(8):800–810, 2001. 7
- [7] M. Donoser, M. Urschler, M. Hirzer, and H. Bischof. Saliency driven total variation segmentation. In *Proc. of IEEE Int. Conf. on Comp. Vis.*, pages 817–824, 2009. 7
- [8] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, 2004. 7
- [9] N. Goyette, P. Jodoin, F. Porikli, J. Konrad, and P. Ishwar. Changedetection.net: A new change detection benchmark dataset. In *IEEE Conf. on Comp. Vis. and Pat. Rec. Workshops*, pages 1–8, 2012. 7
- [10] C. Grana, D. Borghesani, and R. Cucchiara. Optimized block-based connected components labeling with decision trees. *IEEE Trans. on Img. Proc.*, 19(6):1596–1609, 2010. 1
- [11] L. H., Y. C., and K. Suzuki. A run-based two-scan labeling algorithm. *IEEE Trans. on Img. Proc.*, 17(5):749–756, 2008. 1
- [12] R. Haralick and L. Shapiro. *Computer and Robot Vision*, volume 1. Addison-Wesley, 1992. 1
- [13] Z. Li, X.-M. Wu, and S.-F. Chang. Segmentation using superpixels: A bipartite graph partitioning approach. In *Proc. of IEEE Conf. on Comp. Vis. and Pat. Rec.*, pages 789–796, 2012. 7
- [14] E. Mandler and M. Oberlander. One-pass encoding of connected components in multivalued images. In *Proc. of IEEE Int. Conf. on Pat. Rec.*, volume ii, pages 64–69, 1990. 1
- [15] M. Maresca, H. Li, and M. Lavin. Connected component labeling on polymorphic torus architecture. In *Proc. of IEEE Conf. on Comp. Vis. and Pat. Rec.*, pages 951–956, 1988. 1
- [16] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. of IEEE Int. Conf. on Comp. Vis.*, volume 2, pages 416–423, July 2001. 5
- [17] H. Mobahi, S. Rao, A. Yang, S. Sastry, and Y. Ma. Segmentation of natural images by texture and boundary compression. *Int. J. Comput. Vision*, 95(1):86–98, 2011. 7
- [18] A. Rosenfeld and A. C. Kak. *Digital Picture Processing*, volume 2. Academic Press, 1982. 1
- [19] A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *Jnl. of the ACM*, 13(4):471–494, 1966. 1
- [20] H. Sang, J. Zhang, and T. Zhang. Efficient multi-value connected component labeling algorithm and its asic design. *Proc. of SPIE*, 6789:67892I–67892I–8, 2007. 1
- [21] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. on Pat. Anal. and Mach. Intelli.*, 22(8):888–905, 2000. 7

- [22] K. Suzuki, I. Horiba, and N. Sugie. Linear-time connected-component labeling based on sequential local operations. *Comp. Vis. and Img. Understand.*, 89(1):1 – 23, 2003. [1](#)
- [23] J. Wang, Y. Jia, X.-S. Hua, C. Zhang, and L. Quan. Normalized tree partitioning for image segmentation. In *Proc. of IEEE Conf. on Comp. Vis. and Pat. Rec.*, pages 1–8, 2008. [7](#)
- [24] K. Wu, E. Otoo, and K. Suzuki. Optimizing two-pass connected-component labeling algorithms. *Pat. Anal. and App.*, 12(2):117–135, 2009. [1](#)